Računarska gimnazija

Beograd, Knez Mihailova 6

MATURSKI RAD IZ PROGRAMIRANJA I PROGRAMSKIH JEZIKA

RAZVOJ SOFTVERA U OBLAKU

Mentor:

Vladimir Kuzmanović

Učenik:

Tamara Katarić, IV-2

Beograd, maj 2022.

Sadržaj

Uvod

Agencija DARPA (Defense Advanced Research Projects Agency) tražila je 1963. godine od Instituta za tehnologiju u Masačusetsu (MIT) da uz pomoć dva miliona dolara razvije tehnologiju koja bi omogućila da više ljudi istovremeno koristi jedan računar. Jedan od tih računara predstavljao je primitivni oblak kome su mogle da pristupe samo dve ili tri osobe. Takođe, taj računar je predak onome što danas nazivamo računarstvom u oblaku (cloud computing).

J. C. R. Liklider učestvovao je 1963. u razvoju mreže ARPANET (Advanced Research Projects Network), izrazito primitivne verzije interneta. On je promovisao ideju o „Intergalaktičkoj računarskoj mreži“ (Intergalactic Computer Network) koja bi povezivala sve ljude na svetu preko računara i koja bi omogućila pristup informacijama sa bilo koje lokacije. Ta intergalaktička mreža, tj. internet, neophodan je za pristup oblaku.

Na početku se termin „oblak“ odnosio na „prazninu“ koja se nalazila između provajdera i korisnika. Međutim, 1997. godine univerzitetski profesor Ramnat Čelapa sa Univerziteta Emori (Ramnath Chellapa of Emory University) definisao je oblak kao „novu paradigmu računarstva čije će granice biti određene ekonomskim uslovima, a ne isključivo tehničkim aspektima“. Već 1999. godine kompanija Salesforce postala je prva kompanija koja je nudila softver preko interneta, tj. softver kao uslugu (Software as a Service, SaaS).

Godine 2002. Amazon je pokrenuo svoju veb-prodavnicu, a 2006. pokrenuo je i Amazon Web Services koji pruža razne onlajn usluge drugim veb stranicama i klijentima poput skladištenja podataka, izračunavanja i iznajmljivanja virtualnih mašina. Iste godine Gugl (Google) je na tržište izbacio Google Docs, program koji omogućava pravljenje, menjanje i deljenje tekstualnih dokumenata putem interneta. Zatim je 2007. godine Netfliks (Netflix) pokrenuo striming video servis (streaming video service) pomoću oblaka.

Godine 2010. kompanije poput Majkrosofta (Microsoft), Amazona (Amazon) i OpenStack-a razvile su privatne oblake, jer je u javnom oblaku bezbednost bila upitna. Zatim, 2011. IBM je napravio IBM SmartCloud, a Apple je pokrenuo iCloud čiji je fokus bio na svakodnevnim fajlovima poput slika, video snimaka i muzike. Te iste godine, Majkrosoft je počeo putem televizijskih reklama da promoviše oblak i njegove sposobnosti da čuva slike ili video snimke tako da im se može pristupiti bilo kad i sa bilo kog uređaja dokle god postoji internet konekcija.

Godine 2012. Oracle je predstavio Oracle Cloud koji je ponudio sve tri osnovne vrste usluga preko oblaka: infrastrukturu kao uslugu (Infrastructure as a Service, IaaS), platformu kao uslugu (Platform as a Service, PaaS) i softver kao uslugu (Software as a Service, SaaS). Ubrzo su mnoge kompanije počele da nude sve tri ili samo jednu, dok je softver kao usluga postao vrlo popularan.

Kako se oblak dalje razvijao, do 2014. godine bezbednost je postala izrazito značajna. U poslednjih par godina bezbednost informacija u klaudu je umnogome napredovala, iako je pitanje zaštite važnih informacija od slučajnog brisanja ili krađe i dalje od suštinskog značaja za većinu korisnika oblaka. Uprkos tome, danas svi mi na neki način koristimo oblak.

Sve više i više programera koristi oblak u procesu razvijanja softvera. Sve više klaud kompanija razvija alate koji bi olakšali i ubrzali rad programerima. U ovom radu neću se baviti konkretnim alatima za razvoj softvera u oblaku, već ću objasniti nekoliko različitih ideja i principa organizacije koda koje su moguće kada se aplikacija razvija u oblaku.

Razvoj softvera u oblaku ili tradicionalni razvoj

S obzirom da razvoj softvera u oblaku sve više napreduje i postaje sve popularniji, postavlja se pitanje da li je tradicionalni način razvoja softvera zastareo te ga treba zaboraviti i baviti se isključivo budućnošću, tj. razvojem u oblaku. Klaud zaista jeste brži i agilniji, ali to ne mora da znači da treba zaboraviti na tradicionalni način razvoja softvera. Kao i kod većine stvari, nije bitno koji pristup je objektivno bolji, već šta bolje pristaje nekom konkretnom projektu.

Računarstvo u oblaku (cloud computing) podrazumeva isporuku IT resursa po potrebi. To znači da se skladištu, bazama podataka i računarskim kapacitetima pristupa preko provajdera oblaka i da se plaća samo ono sto se koristi. Sa druge strane, tradicionalni pristup podrazumeva da se svi podaci čuvaju lokalno, što znači da je minimalna količina prostora ograničena i određena pre početka razvojnog procesa pa se može desiti da bude skuplje nego što je neophodno.

Razvoj baziran na oblaku (claud-native development) podrazumeva da se kod piše na mašini koja je direktno povezana sa klaud okruženjem (cloud environment) i da se taj kod na jednostavan način prenosi u klaud okruženje. Na taj način omogućava se i drugim programerima da testiraju i unapređuju kod u isto vreme. Samim tim ceo proces razvoja softvera postaje fleksibilniji i brži.

Prilikom razvoja softvera oblak olakšava DevOps (prakse i alati koji omogućuju bržu izradu i isporučivanje aplikacija i servisa) time što omogućava većem broju programera, alata i procesa da rade zajedno u isto vreme, jer se ceo kod čuva na istom mestu. Takođe, osim što je klaud često brži po pitanju pristupa podacima, izrazito je jednostavno povećati skladištene kapacitete što omogućava proširivanje programa u nedogled, bez pogoršavanja performansi. Pored toga, održavanje infrastrukture je mnogo lakše i ažuriranje programa ne podrazumeva privremeni prestanak rada programa kako bi se implementirale sve promene (continuous integration and continuous delivery/deployment, CI/CD).

Kao što je prethodno pomenuto, jedna od glavnih mana oblaka je bezbednost podataka. Korisnicima jeste zabranjen pristup osetljivim podacima, ali zlonamerni napadi su

i dalje mogući pa treba biti oprezan. Neophodno je imati dobre sigurnosne protokole kako bi osetljivi podaci bili zaštićeni od neovlašćenog korišćenja.

Još nešto na šta treba obratiti pažnju je to što provajder oblaka određuje njegovu konfiguraciju (dostupna količina memorije, broj procesora, protok (bandwidth) i sl.) što znači da se ona ne može promeniti, tako da se može desiti da ne odgovara potrebama projekta, što bi dovelo do toga da mora da se traži novi provajder ili da se menja projekat kako bi mu zadata konfiguracija odgovarala.

Kada se koristi tradicionalni pristup, razvojni proces je sekvencijalan. To znači da je lakše držati se razvojnog plana i upravljanje razvojnim procesom je mnogo jednostavnije, dok u oblaku može da bude haotično, jer svi rade istovremeno. Takođe, tradicionalni način razvoja softvera podrazumeva da se svaki aspekt projekta definiše unapred, što znači da je klijent veoma uključen u razvojni proces, ne samo na početku, već i tokom razvoja kako bi mogao da skrene pažnju programerima na ono što je najbitnije. Ukoliko klijent ima specifičan skup zahteva, tradicionalni pristup je bolji od oblaka.

Jedna od glavnih prednosti tradicionalnog pristupa je bezbednost podataka. Podacima se jedino može pristupiti preko uređaja koji skladište te podatke. Samim tim se hakerima otežava posao i podaci su sigurniji.

Ako je cilj da se veliki broj korisnika služi softverom, to često podrazumeva ažuriranje odn. ili proširenje kapaciteta programa, jer se povećava broj korisnika, ili unapređenje rada ili mogućih funkcija programa. Ako je on razvijen na tradicionalni način, korisnici će morati redovno da preuzimaju i instaliraju nova ažuriranja. Takođe, velika je verovatnoća da će neki korisnici nastaviti da koriste starije verzije, što može dovesti do problema sa nekompatibilnošću.

Osim što je brzina razvoja softvera znatno veća kada se radi u oblaku, još jedna važna razlika je mogućnost izmene softvera. Kada se softver razvija na tradicionalan način, on se čuva na više uređaja. To znači da je potrebno mnogo više vremena i napora da se on izmeni, ažurira ili unapredi.

Arhitektura razvoja softvera u oblaku

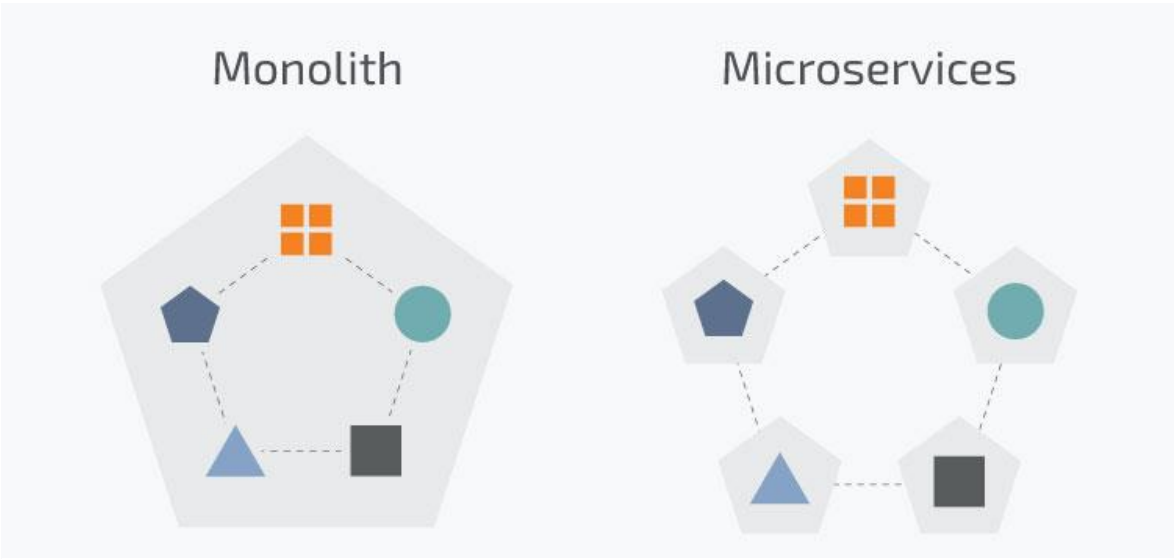
Softver se može organizovati na više načina. Svaki ima prednosti i mane i važno je izabrati onaj koji najbolje odgovara potrebama konkretnog projekta. Najjednostavnija arhitektura je monolitna. Ona podrazumeva jednu celinu, bez nezavisnih komponenti i to znači da se sve radi na nivou cele aplikacije.

Arhitektura orijentisana na servis podrazumeva poboljšanje komunikacije u okviru kompanije. To znači da se servisi od jedne aplikacije mogu lakše i brže nego pre integrisati u novu aplikaciju, ali i da različita odeljenja kompanije mogu efikasnije da komuniciraju jedno sa drugim što povećava brzinu rada i poboljšava iskustvo korisnika.

Mikroservisna aplikacija je podeljena na više delova tj. usluga koje međusobno komuniciraju pomoću API-ja. To znači da se sve radi samo na nivou konkretne usluge, što omogućava veću fleksibilnost i olakšava razumevanje celokupne aplikacije. Nije moguće testirati ili aktivirati celu aplikaciju odjednom i nije moguće uneti promenu na nivou cele aplikacije.

Monoliti (monoliths)

Monolitna arhitektura smatra se tradicionalnim načinom razvoja softvera. Monolitna aplikacija napravljena je tako da se ne može podeliti na više manjih jedinica. Uglavnom se sastoji iz korisničkog interfejsa (client-side user interface), aplikacije sa serverske strane (server-side application) i baze podataka (database) (Slika 1). Glavna karakteristika



Slika 1: Organizacija monolitnih i mikroservisnih aplikacija

monolitne aplikacije je to da je sve ujedinjeno pa se svim funkcijama upravlja sa istog mesta i opslužuju se sa istog mesta, jer je ceo kod napisan na jednom mestu. Zato, kada programeri nešto menjaju u kodu, oni to rade odjednom, u celom projektu.

Jedna od prednosti monolitne arhitekture je da su sveobuhvatne funkcije poput logovanja (logging), rukovanja (handling), keširanja (caching) i praćenja performansi (performace monitoring) lakše za korišćenje i održavanje, jer su vezane samo za jednu aplikaciju. Pošto je monolitni program nedeljiva celina, celokupno testiranje koda (end-to-end testing) je relativno lako i brzo. Takođe, aktivacija (deployment) je lakša, jer se radi o samo jednom fajlu ili direktorijumu. S obzirom da je monolitni pristup razvoju softvera standard, svaki softverski inženjer u stanju je da napravi monolitnu aplikaciju.

Međutim, kada se monolitna aplikacija dalje razvija, može biti toliko velika i komplikovana da postane teška za razumevanje i upravljanje. Što se aplikacija više širi i povećava, to je teže implementirati promene u kodu, jer svaka promena utiče na ceo kod, pa sve izmene moraju biti dobro isplanirane i koordinisane, što produžuje ceo razvojni proces. Nije moguće unaprediti pojedinačne komponente nezavisno od ostatka programa, već samo celokupnu aplikaciju. Takođe, ako se u jednom trenutku ustanovi da bi neka druga tehnologija ili okruženje bilo bolje za dalji razvoj, cela aplikacija bi morala da se ponovo napiše.

Monolitna arhitektura je idealno rešenje ukoliko je ideja da se napravi jednostavna aplikacija uz pomoć malog tima. Tačno je da mikroservisna arhitektura postaje sve popularnija, ali ukoliko je ideja da se aplikacija pokrene što je brže moguće i ukoliko nijedan član tima nema neophodna znanja i veštine za izradu mikroservisne aplikacije, definitivno je bolje odlučiti se za monolitnu arhitekturu.

Servisno orjentisana arhitektura (service-oriented architecture, SOA)

Servisno orijentisana arhitektura predstavlja način na koji se softverske komponente mogu ponovo koristiti i povezivati uz pomoć servisnih intefejsa. Servisi koriste uobičajene interfejs standarde i arhitektonske obrasce kako bi mogli da budu uključeni u nove aplikacije. To omogućava programerima da sastavljaju nove aplikacije pomoću servisa umesto da pišu kod ponovo.

Svaki servis u okviru servisno orijentisane arhitekture podrazumeva i kod i podatke neophodne za izvršavanje konkretne diskretne poslovne funkcije kao što su, na primer, provera kreditne kartice, izračunavanje mesečne rate za otplatu duga i sl. Servisni interfejs omogućava labavo povezivanje, što znači da se usluge mogu koristiti sa malo ili potpuno bez poznavanja načina na koji funkcionišu uz smanjenje zavisnosti između aplikacija.

Interfejs predstavlja vezu između pružaoca usluga i korisnika te usluge. Aplikacije iza interfejsa servisa mogu biti napisane u raznim programskim jezicima kao na primer Java, Microsoft .Net, Cobol i dostupne su u obliku paketa softverskih aplikacija koje obezbeđuju dobavljači poput SAP ili u obliku SaaS aplikacije od dobavljača kao što je Salesforce, a mogu biti i *open source* aplikacije.

Usluge koriste standardne mrežne protokole kako bi slale zahteve za čitanjem ili izmenom podataka. U odgovarajućoj fazi razvoja usluge se objavljuju u registru koji pomaže programerima da ih brzo pronađu i iskoriste prilikom sastavljanja novih aplikacija ili poslovnih procesa.

Ove usluge mogu se izrađivati od nule, mada se često kreiraju tako što se koriste funkcije iz zastarelih sistema kao interfejsi. Na taj način arhitektura bez servera predstavlja značajnu fazu u evoluciji razvoja aplikacije i integraciji u proteklih nekoliko decenija. Pre nego što se pojavila SOA kasnih 90tih godina 20. veka, povezivanje aplikacije sa podacima ili funkcionalnostima obavljalo se u drugom sistemu za koji je bila neophodna kompleksna *point to point* integracija. Tu integraciju programeri su morali u potpunosti ili delimično ponovo da kreiraju za svaki novi projekat. Kada se u okviru SOA te funkcije stave na raspolaganje, programeru se omogućuje da jednostavno iskoristi postojeće kapacitete.

Magistrala poslovnih procesa ESB (enterprize service bus) može se posmatrati kao element svake implementacije SOA, a ta dva pojma ponekad se koriste kao sinonimi što izaziva konfuziju. ESB je obrazac pri čemu centralizovana softverska komponenta vrši integraciju sa pozadinskim sistemima i čini te integracije dostupnim kao servisni interfejs za ponovnu upotrebu od strane novih aplikacija.

Kao nova arhitektura u preduzeću, SOA ima mnoge prednosti u poređenju sa prethodno korišćenim arhitekturama. U njih spada veća poslovna agilnost pri čemu je od suštinskog značaja ponovno korišćenje. Efikasno sastavljanje aplikacije iz recikliranih usluga umesto ponovnog pisanja i reintegracije sa svakim novim projektom omogućuje programerima da mnogo brže kreiraju aplikacije kada to zahtevaju nove poslovne

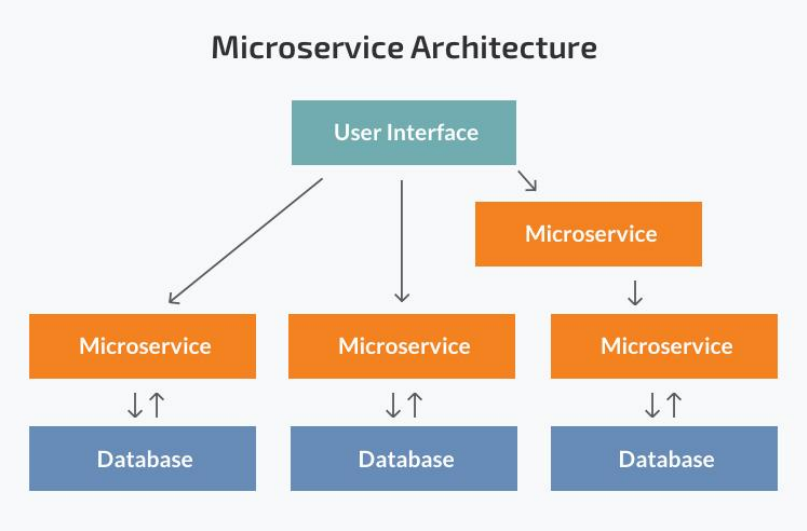
mogućnosti. Takav pristup koji je orijentisan prema uslugama podržava scenarija za integraciju aplikacija, podataka i automatizaciju i orkestriranje usluga u okviru poslovnih procesa ili tokova rada. Na taj način se ubrzava izrada softvera i programeri mnogo manje vremena moraju da ulože u integraciju, a mnogo više mogu da se fokusiraju na unapređenje svojih aplikacija.

U prednosti spada i sposobnost korišćenja starih funkcionalnosti u novim okolnostima: SOA omogućava programerima da na jednostavan način prošire stare funkcionalnosti na novo okruženje. Na primer, mnoge kompanije koriste SOA kako bi iskoristile stare funkcionalnosti iz svojih osnovnih finansijskih sistema u novim aplikacijama i na taj način omogućili svojim klijentima da iskoriste procese i informacije koje su im prethodno bile dostupne samo kroz direktnu interakciju sa zaposlenima ili partnerima kompanije.

Još jednu prednost predstavlja unapređena saradnja između poslovanja i IT sektora: u okviru SOA usluge se mogu definisati kao poslovni pojmovi (npr. „generisanje kvote osiguranja“). Na taj način analitičari poslovanja mogu efikasnije da sarađuju sa programerima kada se radi o značajnim elementima poslovanja kako bi se ostvarili bolji rezultati.

Mikroservisi (microservices)

U okviru mikroservisne arhitekture cela funkcionalnost podeljena je na više manjih, nezavisnih jedinica. Te jedinice predstavljaju neku od funkcionalnosti aplikacije i te jedinice su usluge tj. servisi koji su malih obima. Tako su dobili naziv mikroservisi. Svaka usluga ima sopstvenu bazu podataka i izvršava svoj set funkcija (Slika 2). Usluge međusobno komuniciraju pomoću definisanih metoda tj. API-ja (Application Programming Interfaces). Svaka usluga ima svoj okvir u kome radi i može da se poboljšava, povećava i aktivira nezavisno od svih ostalih.



Slika 2: Šema organizacije mikroservisne aplikacije

Činjenica da su komponente tj. usluge nezavisne jedna od druge, glavna je odlika mikroservisa i podrazumeva da se usluge mogu pojedinačno poboljšavati i aktivirati, što omogućava veću fleksibilnost pri daljem razvoju aplikacije. Takođe, kad dođe do greške ili baga (bug) u kodu on utiče samo na tu konkretnu uslugu, dok ostatak aplikacije nastavlja svoj rad sasvim uobičajeno. To takođe znači da se rizici koji dolaze sa unošenjem bilo kakvih promena smanjuju. Shodno tome, mnogo je lakše dodavati nove funkcionalnosti u aplikaciju kada se koristi mikroservisna nego kada se koristi monolitna arhitektura.

Mikroservisna aplikacija je relativno laka za održavanje, upravljanje i razumevanje, jer je podeljena na više manjih celina. To takođe omogućava svakoj usluzi da se nezavisno razvija što je povoljnije i po pitanju novca i vremena. Zato veliki broj kompanija u nekom trenutku redizajnira svoju aplikaciju i prelazi sa monolitne na mikroservisnu arhitekturu, koja je u stanju da podrži veći broj korisnika i može mnogo lakše da se adaptira ukoliko njihov broj raste.

Nezavisnost jedne usluge od ostalih omogućava da jedan tim od nekoliko programera najčešće bude zadužen za samo jednu uslugu i da ne mora da se drži tehnologije ili okruženja koje je izabrano na početku, već u skladu sa ciljevima koji su postavljeni za neku konkretnu uslugu timovi su u stanju da izaberu najadekvatniju tehnologiju i okruženje.

Glavna mana mikroservisne arhitekture je njena složenost. To znači da se uloge moraju pojedinačno aktivirati, tako da samim tim moraju pojedinačno i da se testiraju. Takođe, neophodno je pažljivo podesiti veze između usluga i baza podataka, a i

sveobuhvatne funkcije poput logovanja i eksterne konfiguracije mogu da budu komplikovane za rešavanje i upravljanje usled složenosti mikroservisne aplikacije.

Mikroservisna arhitektura je idealno rešenje ukoliko je ideja da se razvije velika aplikacija sa više modula i mogućih načina upotrebe (user journeys), ali je važno zapamtiti da je za mikroservisnu aplikaciju potrebno više programera i da je neophodno da bar neko od programera ima konkretna znanja i veštine neophodne za izradu mikroservisne aplikacije.

Arhitektura bez servera

Arhitektura bez servera (serverless architecture ili serverless computing) predstavlja obrazac dizajna softvera koji podrazumeva da provajder oblaka upravlja serverom tj. i softverskim i hardverskim delovima. U arhitekturi bez servera aplikacije zapravo predstavljaju skup individualnih funkcija koje se pojedinačno pozivaju.

Hostovanje aplikacije na internetu često podrazumeva upravljanje serverskom infrastrukturom. To uglavnom znači da je neophodno održavanje i upravljanje virtuelnim ili fizičkim serverom, ali i operativnim sistemom i ostalim procesima hostovanja veb servera koji su neophodni za pokretanje aplikacije. Upotrebom virtuelnog servera pomoću klaud provajdera poput Amazona ili Majkrosofta olakšava se to održavanje, jer provajder upravlja i održava hardver.

Međutim, arhitektura bez servera omogućava programerima da se koncentrišu isključivo na individualne funkcije u kodu aplikacije. Usluge poput AWS Lambda i Microsoft Azure Functions upravljaju i održavaju kako sofverski tako i hardverski deo serverske infrastrukture.

Platforma kao usluga (Platform as a Service, PaaS) nudi neke od istih prednosti kao arhitektura bez servera, često poznata kao funkcija kao usluga (Function as a Service, FaaS). U oba slučaja programeri ne moraju da brinu ni o hardverskim ni o softverskim delovima serverske infrastrukture. Glavna razlika između FaaS i PaaS je u načinu organizacije aplikacije.

Ukoliko se koristi PaaS, aplikacija se aktivira u celini i može da se poveća i širi takođe samo u celini. Razvija se uglavnom u okruženju (framework) poput ASP.NET, Flask, Ruby on Rails i Java Servlets. Moguće je pozvati aplikaciju više puta ukoliko je potrebno.

Ukoliko se koristi FaaS, aplikacija je sačinjena od više autonomnih funkcija koje FaaS provajder pojedinačno hostuje i mogu se povećavati automatski u slučaju da počnu češće da se pozivaju. Takođe, naplaćuje se svaki pojedinačni poziv umesto da se naplaćuje konstantna spremnost aplikacije dok ona čeka da se neka funkcija pozove. Arhitektura bez servera je odlično rešenje za aplikacije koje imaju malo funkcija koje bi FaaS provajder hostovao.

Zaključak

Ideja o internetu postojala je odavno i na samom početku je bila poznata kao intergalaktička računarska mreža, dok se termin „oblak“ odnosio na ono što nismo mogli jasno da definišemo i objasnimo tj. na ono što se nalazi između provajdera i korisnika sve dok ga jedan univerzitetski profesor nije definisao kao novu paradigmu računarstva. On je već 1997. godine predvideo ogroman potencijal za dalji razvoj, a već samo dve godine kasnije pojavila se prva kompanija koja je nudila SaaS. Kompanije koje su vremenom promenile svet i imale kulturološki uticaj, kao što su Amazon, Gugl i Netfliks, početkom 21. veka pokrenule su na tržištu usluge koje se i dan danas koriste.

U drugoj deceniji 21. veka počinju da se razvijaju različite vrste oblaka poput privatnog i sve se više pažnje posvećuje zaštiti podataka u oblaku. Takođe, Majkrosoft promoviše oblak i njegovu sposobnost da čuva svakodnevne fajlove poput slika, video snimaka i muzike tako da im se može pristupiti bilo kad i sa bilo kog uređaja pod uslovom da je povezan na internet.

Razvoj softvera u oblaku omogućava da veći broj programera radi na programu u isto vreme. To znatno ubrzava ceo razvojni proces. Međutim, ako je izrazito važno držati se razvojnog plana ili ako klijent ima specifičan skup zahteva, razvoj softvera na tradicionalan način može biti bolje rešenje od oblaka. Klaud omogućava lakšu izmenu koda i ažuriranje aplikacije nego tradicionalni način. Takođe, jedna od glavnih prednosti razvoja u klaudu je to što je često povoljniji, odn. plaća se samo ono što se koristi, a kako se menjaju potrebe aplikacije tako se menja i cena.

Monolitne aplikacije su organizovane kao jedna celina, bez nezavisnih delova. To znači da kada se unose promene, unose se na nivou cele aplikacije, kada se aplikacija testira ili aktivira ona se cela testira ili aktivira odjednom, u celini, vezana je za samo jednu bazu podataka i ukoliko u bilo kom delu aplikacije dođe do bilo kakve greške, cela aplikacija prestaje sa radom dok programer ne reši problem. Takođe, kako se aplikacija dalje razvija i povećava, postaje sve komplikovanija za razumevanje i upravljanje, što otežava dalje unošenje promena i time usporava i otežava razvojni proces. Monolitna arhitektura je dobro rešenje za male aplikacije.

Arhitektura orijentisana na usluge slična je mikroservisnoj arhitekturi. Ključna razlika leži u obimu poslovanja za koji se koriste ove dve arhitekture. Mikroservisna

arhitektura odnosi se na samo jednu aplikaciju, a arhitektura orijentisana na usluge odnosi se na celu kompaniju koja ima više aplikacija. SOA omogućava da se servis koji je potreban većem broju aplikacija napiše samo jednom i da se onda lako i jednostavno integriše sa svim aplikacijama kojima je potreban umesto da se piše ponovo za svaku. Na taj način softver postaje fleksibilniji i agilniji. Takođe poboljšava komunikaciju različitih delova kompanije.

Mikroservisne aplikacije podeljene su na manje delove tj. pojedinačne usluge ili funkcionalnosti. Usluge međusobno komuniciraju pomocu API-ja i svaka od njih se nezavisno od ostalih razvija, testira i aktivira i ima svoju bazu podataka pa je u potpunosti nezavisna od ostatka aplikacije. Ako dođe do bilo kakve greške u aplikaciji, ona utiče samo na konkretnu uslugu dok ostatak aplikacije nastavlja svoj rad kao i obično. Lakše je širiti mikroservisnu aplikaciju od monolitne, jer su sve usluge nezavisne pa je i lakša za razumevanje i jedan tim od nekoliko programera zadužen je najčešće za samo jednu uslugu. Međutim, ta podeljenost aplikacije na manje, nezavisne, celine znatno povećava njenu složenost u odnosu na monolitnu aplikaciju. Iz tog razloga mikroservisna arhitektura je dobro rešenje za veće aplikacije koje treba da nastave da se šire i imaju veliki broj funkcionalnosti. Za manje aplikacije mikroservisna arhitektura je suviše kompleksno rešenje koje se ne bi isplatilo.

Arhitektura bez servera podrazumeva autsorsovanje serverske infrastrukture. Poznata je i kao FaaS. Provajderi poput Amazona i Majkrosofta imaju svoje servise koji upravljaju i održavaju softverski i hardverski deo serverske infrastrukture što znači da programeri mogu samo da se koncentrišu na pisanje koda. Aplikacija u arhitekturi bez servera predstavlja skup individualnih funkcija koje se pojedinačno pozivaju. To znači da se pojedinačno i razvijaju. Razlika između FaaS-a i PaaS-a je sledeća: ukoliko se koristi PaaS aplikacija predstavlja jednu celinu što znači da se aktivira u celini, proširuje se samo kao jedna celina i tako se i naplaćuje nezavisno od broja poziva; kada se koristi FaaS naplaćuje se samo svaki pojedinačni poziv funkcije umesto da se naplaćuje spremnost funkcije da primi poziv.

Možemo reći da je izum oblaka promenio svet. Ne samo što ga svi mi koristimo svakog dana u svojim privatnim životima, već je oblak takođe uticao na ekonomiju i rad kompanija i programera. Danas postoji više različitih načina organizacije koda u aplikaciji i više različitih načina organizacije softvera u kompaniji. Poslovni procesi se izvršavaju sve brže i sa sve manje grešaka. Poput programera i preduzetnici su postali svesni da nije svako

rešenje primenjivo u svakoj situaciji. Veliki pozitivan efekat razvoja softvera u oblaku jeste činjenica da sada imamo mnogo više različitih rešenja koja odgovaraju različitim potrebama projekata.

Literatura

https://www.dataversity.net/brief-history-cloud-computing/#

https://solved.scality.com/solved/the-history-of-cloud-computing/

https://intersog.com/blog/cloud-development-vs-traditional-development/

https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

https://www.ibm.com/cloud/learn/soa

https://saga.rs/resenja-i-servisi/enterprise-service-bus/

https://www.twilio.com/docs/glossary/what-is-serverless-architecture

https://www.datadoghq.com/knowledge-center/serverless-architecture/